



알고리즘

[기하] 외적을 이용해서 선분과 선분의 교차점 구하기

bowbowbow 2016.03.27 22:19

목차

[기하]외적을 이용해서 선분과 선분의 교차점 구하기

직선과 직선의 교차점

선분과 선분의 교차점

선분과 선분의 교차여부 판별

References

이 글은 포스팅 [기하]외적을 이용한 두 벡터의 상대적인 방향 판별에 설명 되어있는 배경지식을 전제로 합니다. 또 이 글은 평면에서의 선분과 선분의 교차점에 대해 설명합니다.

계산 기하 문제에서 **선분** 은 기하를 구성하는 **기본 요소** 이기 때문에 선분의 교차점을 구하는 것은 기하문제를 풀 때 매우 자주 등장합니다.

그러나 선분과 선분의 교차점을 구하는 것은 쉽지 않습니다. 교차점을 구하는 것은 고등학교1학년 수학책에 나오는 내용으로 개념 자체는 어렵지 않지만 모든 경우에 대한 일반해를 프로그래밍적으로 구하는 것이 까다롭기 때문입니다.

직선과 직선의 교차점

선분은 **직선의 일부분** 이기 때문에 선분과 선분에 대한 교차점을 논의하기 전에 먼저 **직선과 직선의 교차점** 을 구해보시다.



어떻게 직선과 직선의 교차점에 대한 간결한 코드를 작성할 수 있을까요? 두 직선을 벡터로 표현 하고 벡터의 연산을 이용해서 연립방정식을 풀면 외적으로 표현된 간결한 해를 얻을 수 있습니다.

두 직선을 다음과 같이 벡터로 표현합니다. $\vec{a} + p\vec{b}$, $\vec{c} + q\vec{d}$
(이때 \vec{b} 와 \vec{d} 는 각 직선에 평행한 벡터입니다. 직선을 벡터로 표현하는 것에 대한 이해가 필요합니다.)

그 다음 각 직선 위의 점은 성분 표현으로 $(a_x + pb_x, a_y, pb_y)$, $(c_x + qd_x, c_y, qd_y)$ 으로 나타낼 수 있고 아래 연립방정식을 얻을 수 있습니다.

$$a_x + pb_x = c_x + qd_x$$

$$a_y + pb_y = c_y + qd_y$$

이 연립방정식을 p에 대해 풀면 다음과 같습니다.

$$p = \frac{(c_x - a_x)d_y - (c_y - a_y)d_x}{b_xd_y - b_yd_x}$$

이때 분모가 0인 경우는 두 직선이 평행한 경우로 교점이 정의되지 않습니다. 따라서 이 경우를 제외하고 생각합니다.

이제 p를 외적으로 더 간략하게 나타낼 수 있습니다.

$$p = \frac{(\vec{c} - \vec{a}) \times \vec{d}}{\vec{b} \times \vec{d}} \quad (\vec{b} \times \vec{d} \neq 0)$$

이제 위 개념을 바탕으로 직선과 직선의 교차점을 구하는 코드를 작성하겠습니다.

벡터를 표현하는 구조체 `vector2`를 만들고 두 직선의 교차점을 반환하는 함수 `lineIntersection`를 만들었습니다.

```
/* 알고리즘 문제 해결전략 1권, 구종만, 인사이트(2012), Capter15 계산기하 소스 */
```

```
#define EPSILON 0.0001
```

```
// 먼저 벡터를 표현하는 구조체를 정의합니다.
```

```
struct vector2{
```

```
    double x, y;
```

```
    //생성자
```

```
    vector2(double _x, double _y){
```

```
        x = _x, y = _y;
```

```
    }
```

```
    //외적
```

```
    double cross(const vector2& other) const{
```

```
        return x*other.y-y*other.x;
```

```
    }
```

```
/* 연산자 오버로딩을 통해 실제 벡터의 연산을 구현합니다. */
```



bowbowbow

멍멍멍

CATEGORY

```

// 벡터의 덧셈
vector2 operator + (vector2 other) const{
    return vector2(x + other.x, y + other.y);
}
// 벡터의 뺄셈
vector2 operator - (vector2 other) const{
    return vector2(x - other.x, y - other.y);
}
// 두 벡터의 비교
bool operator == (vector2 other) const{
    return x == other.x && y == other.y;
}
bool operator < (vector2 other) const{
    return x < other.x && y < other.y;
}
};

// - 점 a, b를 지나는 직선과 점 c, d를 지나는 직선의 교점을 x에 반환한다.
// - 두 직선이 평행이면(겹치는 경우 포함) 거짓을, 아니면 참을 반환한다.
bool lineIntersection(vector2 a, vector2 b, vector2 c, vector2 d, vector2& x){
    double det = (b-a).cross(d-c);
    // 두선이 평행인 경우
    if(fabs(det) < EPSILON) return false;
    x = a+(b-a)*((c-a).cross(d-c)/det);
    return true;
}

```

코드에서 **EPSILON**은 매우 작은 실수를 뜻하는 상수입니다. 부동소수점으로 나타내어지는 실수는 정밀도가 떨어지기 때문에 `fabs(det)` 실제 0이더라도 0.00000001같은 값을 가질 수 있습니다. 따라서 `fabs(det)`가 0인지를 판단하기 위해 `fabs(det) < EPSILON`을 사용합니다.

EPSILON은 그리스 스무번째 글자입니다. 이 문자가 매우 작은 수라는 뜻을 가지게 된 것은 함수의 극한을 엄밀하게 정의할때 사용하는 **입실론-델타 논법**에서 유래되었습니다. 수학자들은 배가 엄청 부를 때 “이제 입실론 만큼 더 먹을 수 있겠다”고 한다고 합니다...

선분과 선분의 교차점

저는 위에서 직선과 직선의 교차점을 구할 수 있으면 선분과 선분의 교차점 구하는건 매우 간단하다고 생각했습니다. 교차점이 모두 선분위에 오는지만 확인하면 되기 때문이죠. 그러나 **두 선분이 모두 한 직선위에 있는 경우**에 처리하기가 까다롭습니다.

한 직선 위에 두 선분이 있을 때 이 선분들의 관계는 넷 중 하나입니다.

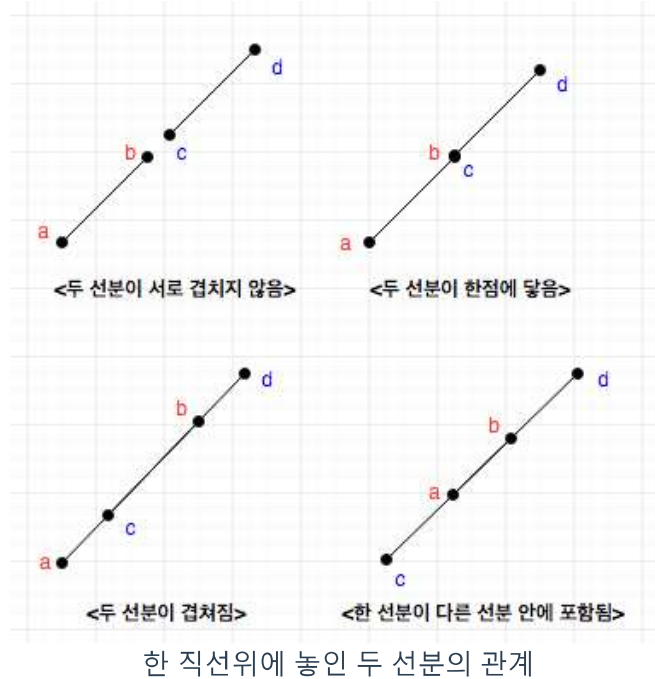


bowbowbow

명명명

CATEGORY

- 한 선분이 다른 선분 안에 포함됨



위에서 정의한 **lineIntersection** 함수는 위 네가지 경우 모두에 대해 거짓을 반환하는데 2, 3, 4 경우는 교차한다고 말할 수 있기 때문입니다. 따라서 이 들까지 모두 고려하는 구현을 해야합니다.

아래 구현에서 선분의 교차점을 구하는 함수 **segmentIntersection** 를 새로 정의하고 이전에 정의한 직선의 교차점을 구하는 함수 **lineIntersection** 를 활용해서 선분을 직선이라고 생각하고 직선의 교차점을 구합니다. 이 때 두 선분이 평행하면 위 네가지 경우 즉 두 선분이 모두 한직선위에 있는 경우를 처리하는 함수인 **paralleSegments** 를 호출합니다. 아니면 **lineIntersection** 함수가 반환한 교차점이 두 선분위에 있는 점인지를 판별하기 위해 **inBoundingRectangle** 함수를 정의하고 사용합니다.

구현에서 **vector2** 구조체 가 아주 유용하게 사용됩니다. 단순한 뺄셈, 덧셈, 비교 연산을 이 구조체를 통해 아주 쉽게 처리할 수 있습니다. 만약 **vector2** 구조체가 없었다면 구현이 간결하지 못했을 것입니다.

```
/* 알고리즘 문제 해결전략 1권, 구종만, 인사이트(2012), Capter15 계산기하 소스 */

//원점에서 벡터 b가 벡터 a의 반시계 방향이면 양수, 시계방향이면 음수, 평행이면 0을 반환 한다.
double ccw(vector2 a, vector2 b){
    return a.cross(b);
}

//점 p를 기준으로 벡터 b가 벡터 a의 반시계 방향이면 양수, 시계방향이면 음수, 평행이면 0을 반환 한다.
double ccw(vector2 p, vector2 a, vector2 b){
    return ccw(a-p, b-p);
}
```



bowbowbow

멍멍멍

CATEGORY

```

    if(d < c) swap(c,d);

    //한 직선위에 없거나 두 선분이 겹치지 않는 경우를 우선 걸러낸다. 본문의 1번 관계인 경우이다.
    if(ccw(a, b, c) != 0 || b < c || d < a) return false;

    //두 선분이 확실히 겹친다면 교차점 하나를 찾는다.
    if(a<c) p = c;
    else p = a;
    return true;
}

// - p가 두 점 a, b를 감싸면서 각 변이 x, y축에 평행한 최소사각형 내부에 있는지 확인한다.
// a, b, p는 일직선 상에 있다고 가정한다.
bool inBoundingRectangle(vector2 p, vector2 a, vector2 b){
    if(b < a) swap(a, b);
    return p == a || p == b || (a < p && p < b);
}

// - 두 점 a, b를 지나는 선분과 두 점 c, d를 지나는 선분을 p에 반환한다.
// - 교점이 여러개일 경우 아무점이나 반환한다.
bool segmentIntersection(vector2 a, vector2 b, vector2 c, vector2 d, vector2& p)
{
    //두 직선이 평행인 경우를 우선 예외로 처리한다.
    if(!lineIntersection(a, b, c, d, p))
        return paralleSegments(a, b, c, d, p);
    //p가 두 선분에 포함되어 있는 경우에만 참을 반환한다.
    return inBoundingRectangle(p, a, b) && inBoundingRectangle(p, c, d);
}

```

앞의 헤더파일을 포함해서 두 선분의 교차점 구하는 전체 소스입니다.

[소스 보기](#)

선분과 선분의 교차여부 판별

교차점을 구할필요없이 두선분의 교차여부만 판별해야하는 상황이라면 ccw()함수를 이용해서 단순하게 구할 수 있습니다. ccw 함수는 아래 두 함수로 [기하]외적을 이용한 두 벡터의 상대적인 방향 판별에 자세히 설명되어있습니다.

```

//원점에서 벡터 b가 벡터 a의 반시계 방향이면 양수, 시계방향이면 음수, 평행이면 0을 반환 한다.
double ccw(vector2 a, vector2 b){

```



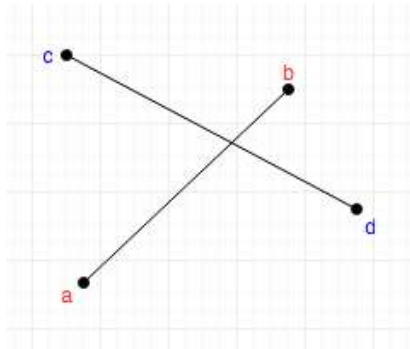
bowbowbow

멍멍멍

CATEGORY

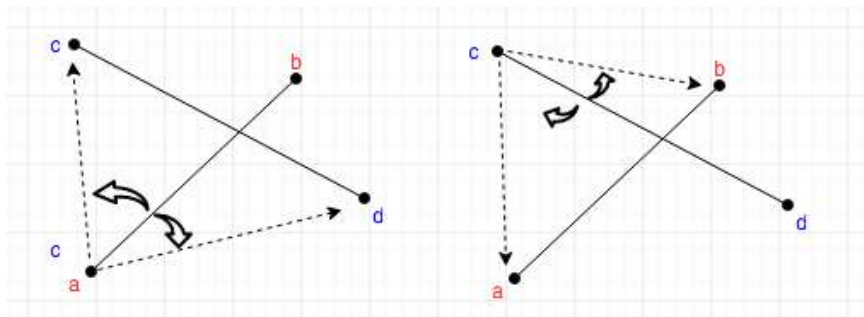
```
double ccw(vector2 p, vector2 a, vector2 b){
    return ccw(a-p, b-p);
}
```

아이디어는 그림을 보면 직관적으로 알 수 있습니다. 아래 그림 처럼 교차하는 두 선분 \overline{ab} 과 \overline{cd} 이 있다고 합시다.



교차하는 두 선분

이때 a 입장에서 보면 c와 d 중 한쪽은 b를 기준으로 시계방향에, 다른 한쪽은 시계반대방향에 있습니다. 반대로 c입장에서 보면 a와 b중 한쪽은 d의 시계방향에, 다른 한쪽은 시계반대방향에 있어야 합니다.

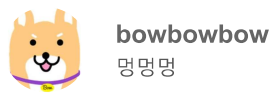


교차하는 두 선분 각도

즉 $ccw(a, b, c)$ 와 $ccw(a, b, d)$ 의 곱이 음수이고 $ccw(c, d, a)$ 와 $ccw(c, d, b)$ 의 곱이 음수여야 합니다. 이를 아래 `segmentIntersects` 함수에서 구현했습니다. 두 선분이 교차하지 않고 접촉하는 경우도 인식하기 위해 ab나 cd중 하나가 0인 경우도 참을 반환하도록 했습니다.

// 두 선분이 서로 접촉하는지 여부를 반환한다.

```
bool segmentIntersects(vector2 a, vector2 b, vector2 c, vector2 d){
    double ab = ccw(a, b, c)*ccw(a, b, d);
    double cd = ccw(c, d, a)*ccw(c, d, b);
    // 두 선분이 한 직선에 위에 있거나 끝점이 겹치는 경우
    if(ab == 0 && cd == 0){
        if(b < a) swap(a, b);
        if(d < c) swap(c, d);
        return !(b < c || d < a);
    }
}
```



CATEGORY

References

알고리즘 문제 해결전략 1권, 구종만, 인사이트(2012)

20

구독하기


좋아요 0개

TAG 계산 기하, 교차 점, 기하, 선분, 외적

댓글 5



구름속의산책
잘보고갑니다
2016.03.27 22:47 [신고](#)



산마음
직선과직선의 교차점 구하기 에대해서 많은도움받고 갑니다.^^
2016.03.28 22:07 [신고](#)



귀향을준비하는486
업무에 많은 도움이 됐습니다. 지식의 나눔에 감사를 ...
2016.07.29 15:16 [신고](#)





김진오

좋은 정보 감사합니다 덕분에 삽질 안하고 잘 해결했습니다

2018.11.25 09:31 [신고](#)

여러분의 소중한 댓글을 입력해주세요

이름

비밀번호

비밀글 입력